

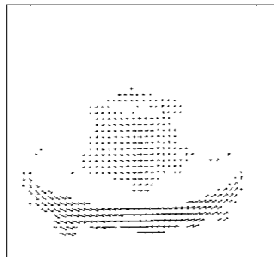
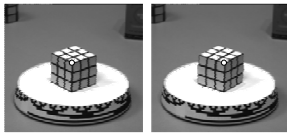
Targil 9 :

Optical Flow

Lecture Outline

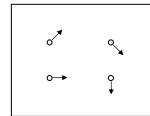
- Defining the optical flow problem
- Patch matching (correlation)
- The aperture problem
- Optical flow equation (gradient descent on the error function)
- Lukas-Kanade flow
- Iterative refinement
- Course to Fine solution (pyramids)
- Multi-resolution Lucas Kanade Algorithm

Motion estimation: Optical flow

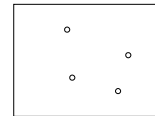


We have seen how to estimate the image of the entire image, now we want to estimating motion of each pixel separately

Problem definition: optical flow



$H(x, y)$



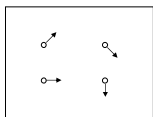
$I(x, y)$

How to estimate pixel motion from image H to image I?

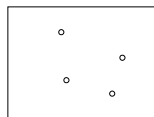
Why is it useful

- Depth (3D) reconstruction
- Motion detection - tracking moving objects
- Compression.
-

Problem definition: optical flow



$H(x, y)$



$I(x, y)$

How to estimate pixel motion from image H to image I?

- Solve pixel correspondence problem
 - given a pixel in H, look for nearby pixels of the same color in I

Key assumptions

- **color constancy**: a point in H looks the same in I
 - For grayscale images, this is **brightness constancy**
- **small motion**: points do not move very far

This is called the **optical flow** problem

Classes of Techniques

Feature-based methods

- Extract visual features (corners, textured areas) and track them over multiple frames
- Sparse motion fields, but possibly robust tracking
- Suitable especially when image motion is large (10-s of pixels)

Direct-methods

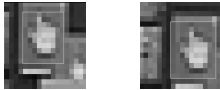
- Directly recover image motion from spatio-temporal image brightness variations
- Motion vectors directly recovered without an intermediate feature motion calculation
- Dense motion fields, but more sensitive to appearance variations
- Suitable for video and when image motion is small (< 10 pixels)

Patch matching (revisited)

How do we determine correspondences?

- *block matching* or *SSD* (sum squared differences)

$$E(x, y; d) = \sum_{(x', y') \in N(x, y)} [I_L(x' + d, y') - I_R(x', y')]^2$$



Matching Criteria - Difference

Common matching criteria:

- **SSD** - Sum of Squared Differences

$$d(p_1, p_2) = \sum_{j=-k}^k \sum_{i=-k}^k (I_1(x_1 + i, y_1 + j) - I_2(x_2 + i, y_2 + j))^2$$

k - the size of the window.

$p_1 = (x_1, y_1)$ is the center of the window in I_1 .

$p_2 = (x_2, y_2)$ is the center of the window in I_2 .

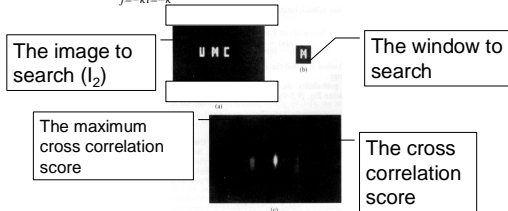
- **SAD** - Sum of Absolute Differences

$$d(p_1, p_2) = \sum_{j=-k}^k \sum_{i=-k}^k |I_1(x_1 + i, y_1 + j) - I_2(x_2 + i, y_2 + j)|$$

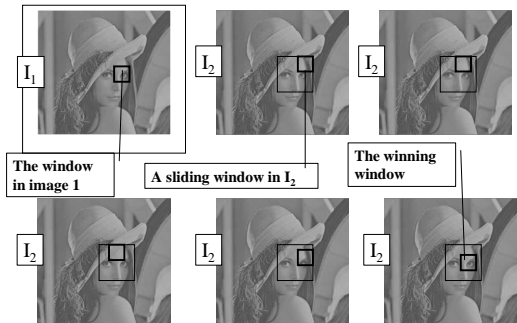
Matching Criteria - Correlation

Max Cross Correlation is similar to Min SSD, but can be implemented more efficiently:

$$d(p_1, p_2) = \sum_{j=-k}^k \sum_{i=-k}^k I_1(x_1 + i, y_1 + j) * I_2(x_2 + i, y_2 + j)$$



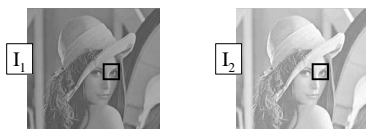
Matching by sliding window



Handling illumination changes

SSD, SAD and Cross Correlation assume constant brightness.

On order to handle illumination changes, Normalized cross-correlation can be used.



Normalized Cross Correlation

When ordering the pixels in the windows in vectors v_1, v_2 :

$$ssd(\vec{v}_1, \vec{v}_2) = \sum_i (\vec{v}_1(i) - \vec{v}_2(i))^2 = \|\vec{v}_1 - \vec{v}_2\|^2 \quad \leftarrow \text{Squared Vector norm}$$

$$corr(\vec{v}_1, \vec{v}_2) = \sum_i \vec{v}_1(i) * \vec{v}_2(i) = \langle \vec{v}_1, \vec{v}_2 \rangle \quad \leftarrow \text{Inner product}$$

The Normalized Cross Correlation is:

$$NorCorr(\vec{v}_1, \vec{v}_2) = \frac{\sum_i \vec{v}_1(i) * \vec{v}_2(i)}{\sqrt{\sum_i \vec{v}_1(i) * \vec{v}_1(i)} \sqrt{\sum_i \vec{v}_2(i) * \vec{v}_2(i)}} = \frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{\|\vec{v}_1\| * \|\vec{v}_2\|}$$

Where we are subtracting the average of the patch from the vector

$$\vec{v}_1 = \vec{v}_1 - \bar{v}_1$$

$$\vec{v}_2 = \vec{v}_2 - \bar{v}_2$$

Hebrew University Image Processing - 2005

Aperture problem

13

Hebrew University Image Processing - 2005

Aperture problem

14

Hebrew University Image Processing - 2005

Aperture problem

15

Hebrew University Image Processing - 2005

Direct methods

gradient descent on the error function

Same assumption we used in finding global motion (image alignment)

16

Hebrew University Image Processing - 2005

Optical flow constraints (grayscale images)

Let's look at these constraints more closely

- brightness constancy: Q: what's the equation?
- small motion: (u and v are less than 1 pixel)
 - suppose we take the Taylor series expansion of I:

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

$$\approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

17

Hebrew University Image Processing - 2005

Optical flow equation

Combining these two equations

$$0 = I(x+u, y+v) - H(x, y) \quad \text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

$$\approx I(x, y) + I_x u + I_y v - H(x, y)$$

$$\approx (I(x, y) - H(x, y)) + I_x u + I_y v$$

$$\approx I_t + I_x u + I_y v$$

$$\approx I_t + \nabla I \cdot [u \ v]$$

In the limit as u and v go to zero, this becomes exact

$$0 = I_t + \nabla I \cdot \left[\frac{\partial x}{\partial t} \ \frac{\partial y}{\partial t} \right]$$

18

Optical flow equation

$$0 = I_t + \nabla I \cdot [u \ v]$$

Q: how many unknowns and equations per pixel?

Intuitively, what does this constraint mean?

- The component of the flow in the gradient direction is determined
- The component of the flow parallel to an edge is unknown

This explains the Barber Pole illusion
<http://www.sandlotscience.com/Ambiguous/barberpole.htm>



The Aperture Problem

$$\text{Let } M = \sum (\nabla I)(\nabla I)^T \quad \text{and} \quad b = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

- Algorithm: At each pixel compute U by solving $MU=b$
- M is singular if all gradient vectors point in the same direction
 - e.g., along an edge
 - of course, trivially singular if the summation is over a single pixel or there is no texture
 - i.e., only *normal flow* is available (aperture problem)
- Corners and textured areas are OK

Getting more Equations

How to get more equations for a pixel?

- Basic idea: impose additional constraints
 - most common is to assume that the flow field is smooth locally
 - one method: pretend the pixel's neighbors have the same (u,v)
 - If we use a 5x5 window, that gives us 25 equations per pixel!

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [u \ v]$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$\underbrace{\hspace{10em}}_A \quad \underbrace{\hspace{2em}}_d \quad \underbrace{\hspace{10em}}_b$
 $25 \times 2 \quad 2 \times 1 \quad 25 \times 1$

RGB version

How to get more equations for a pixel?

- Basic idea: impose additional constraints
 - most common is to assume that the flow field is smooth locally
 - one method: pretend the pixel's neighbors have the same (u,v)
 - If we use a 5x5 window, that gives us 25*3 equations per pixel!

$$0 = I_t(p_i)[0, 1, 2] + \nabla I(p_i)[0, 1, 2] \cdot [u \ v]$$

$$\begin{bmatrix} I_x(p_1)[0] & I_y(p_1)[0] \\ I_x(p_1)[1] & I_y(p_1)[1] \\ I_x(p_1)[2] & I_y(p_1)[2] \\ \vdots & \vdots \\ I_x(p_{25})[0] & I_y(p_{25})[0] \\ I_x(p_{25})[1] & I_y(p_{25})[1] \\ I_x(p_{25})[2] & I_y(p_{25})[2] \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1)[0] \\ I_t(p_1)[1] \\ I_t(p_1)[2] \\ \vdots \\ I_t(p_{25})[0] \\ I_t(p_{25})[1] \\ I_t(p_{25})[2] \end{bmatrix}$$

$\underbrace{\hspace{10em}}_A \quad \underbrace{\hspace{2em}}_d \quad \underbrace{\hspace{10em}}_b$
 $75 \times 2 \quad 2 \times 1 \quad 75 \times 1$

Lukas-Kanade flow

Prob: we have more equations than unknowns

$$\begin{matrix} A & d = b & \longrightarrow & \text{minimize } \|Ad - b\|^2 \\ 25 \times 2 & 2 \times 1 & & 25 \times 1 \end{matrix}$$

Solution: solve least squares problem

- minimum least squares solution given by solution (in d) of:

$$\begin{matrix} (A^T A) & d = A^T b \\ 2 \times 2 & 2 \times 1 & 2 \times 1 \end{matrix}$$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$\underbrace{\hspace{10em}}_{A^T A} \quad \underbrace{\hspace{10em}}_{A^T b}$

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lukas & Kanade (1981)

Conditions for solvability

- Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$\underbrace{\hspace{10em}}_{A^T A} \quad \underbrace{\hspace{10em}}_{A^T b}$

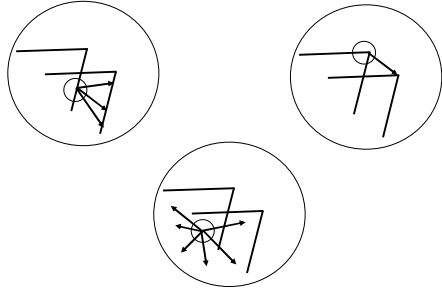
When is This Solvable?

- $A^T A$ should be invertible
- $A^T A$ should not be too small due to noise
 - eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
 - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

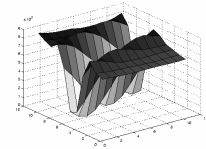
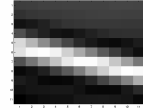
$A^T A$ is solvable when there is no aperture problem

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \ I_y] = \sum \nabla I (\nabla I)^T$$

Local Patch Analysis

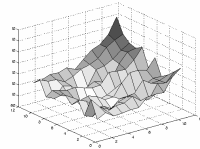
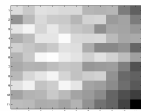


Edge



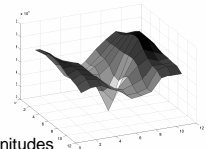
$\sum \nabla I (\nabla I)^T$
 - large gradients, all the same
 - large λ_1 , small λ_2

Low texture region



$\sum \nabla I (\nabla I)^T$
 - gradients have small magnitude
 - small λ_1 , small λ_2

High textured region



$\sum \nabla I (\nabla I)^T$
 - gradients are different, large magnitudes
 - large λ_1 , large λ_2

Observation

This is a two image problem BUT

- Can measure sensitivity by just looking at one of the images!
- This tells us which pixels are easy to track, which are hard
 - very useful later on when we do feature tracking...

Errors in Lukas-Kanade

What are the potential causes of errors in this procedure?

- Suppose $A^T A$ is easily invertible
- Suppose there is not much noise in the image

When our assumptions are violated

- Brightness constancy is **not** satisfied
- The motion is **not** small
- A point does **not** move like its neighbors
 - window size is too large
 - what is the ideal window size?

Limits of the gradient method

Fails when intensity structure in window is poor
 Fails when the displacement is large (typical operating range is motion of 1 pixel)

Linearization of brightness is suitable only for small displacements

Also, brightness is not strictly constant in images

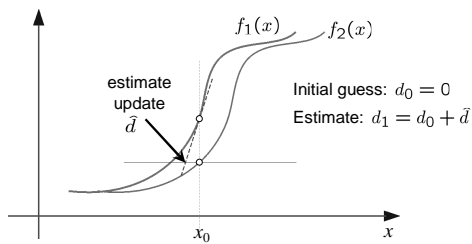
actually less problematic than it appears, since we can pre-filter images to make them look similar

Iterative Refinement

Iterative Lukas-Kanade Algorithm

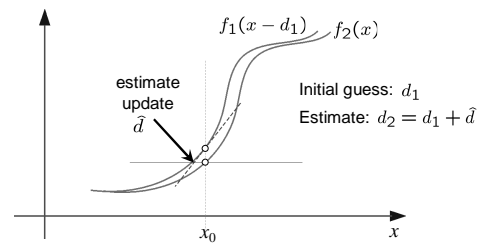
1. Estimate velocity at each pixel by solving Lucas-Kanade equations
2. Warp H towards I using the estimated flow field
 - use image warping techniques (easier said than done)
3. Repeat until convergence

Optical Flow: Iterative Estimation

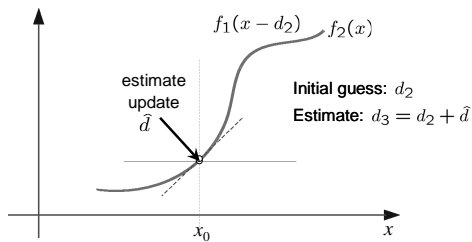


(using d for displacement here instead of u)

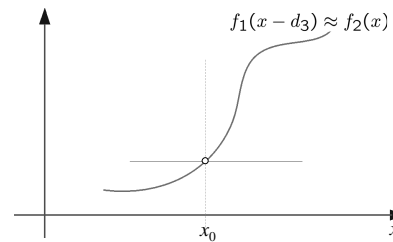
Optical Flow: Iterative Estimation



Optical Flow: Iterative Estimation



Optical Flow: Iterative Estimation



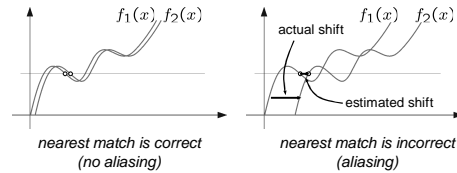
Optical Flow: Iterative Estimation

Some Implementation Issues:

- Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
- Warp one image, take derivatives of the other so you don't need to re-compute the gradient after each iteration.
- Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Optical Flow: Aliasing

Temporal aliasing causes ambiguities in optical flow because images can have many pixels with the same intensity. I.e., how do we know which 'correspondence' is correct?



To overcome aliasing: coarse-to-fine estimation.

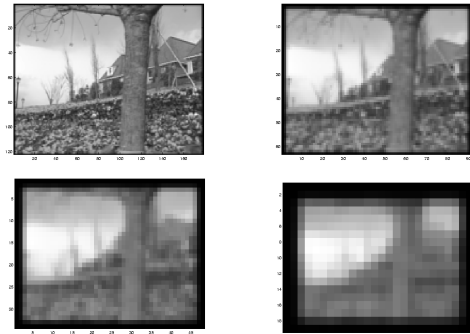
Revisiting the small motion assumption



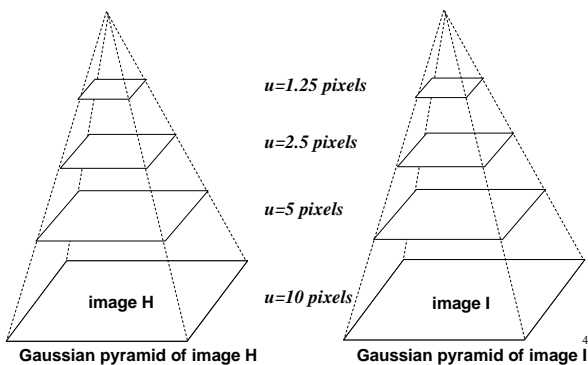
Is this motion small enough?

- Probably not—it's much larger than one pixel (2nd order terms dominate)
- How might we solve this problem?

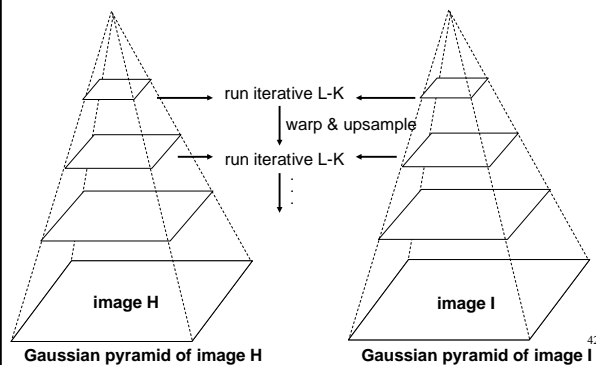
Reduce the resolution!



Coarse-to-fine optical flow estimation



Coarse-to-fine optical flow estimation



Multi-resolution Lucas Kanade Algorithm

Compute Iterative LK at highest level

•For Each Level i

•Take flow $u^{(i-1)}, v^{(i-1)}$ from level $i-1$

•Upsample the flow to create $u^*(i), v^*(i)$ matrices of twice resolution for level i .

•Multiply $u^*(i), v^*(i)$ by 2

•Compute I_i from a block displaced by $u^*(i), v^*(i)$

•Apply LK to get $u'(i), v'(i)$ (the correction in flow)

•Add corrections $u'(i), v'(i)$ to obtain the flow $u(i), v(i)$ at i th level, i.e., $u(i)=u^*(i)+u'(i), v(i)=v^*(i)+v'(i)$

Optical Flow: Iterative Estimation

Some Implementation Issues:

- Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
- Warp one image, take derivatives of the other so you don't need to re-compute the gradient after each iteration.
- Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Beyond Translation

So far, our patch can only translate in (u,v)

What about other motion models?

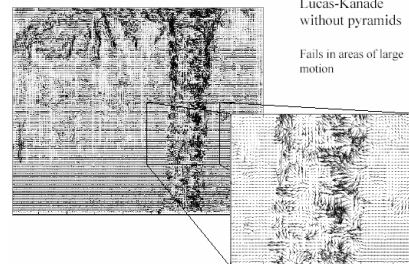
- rotation, affine, perspective

Same thing but need to add an appropriate Jacobian (see Table 2 in Szeliski handout):

$$A^T A = \sum_i J^T I_i (\nabla I)^T J$$

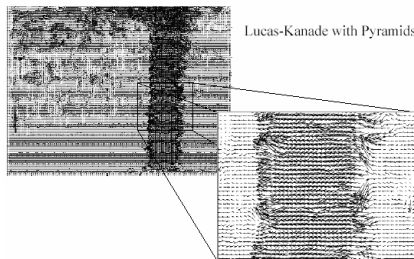
$$A^T b = - \sum_i J^T I_i (\nabla I)^T$$

Optical Flow Results



Lucas-Kanade without pyramids
Fails in areas of large motion

Optical Flow Results



Lucas-Kanade with Pyramids

Optical flow Results

